A Semi-automatic Approach to Identifying and Unifying Ambiguously Encoded Arabic-Based Characters

Sardar Jaf
School of Engineering and Computer Sciences
The University of Durham
Durham, United Kingdom
sardar.jaf@durham.ac.uk

***Abstract*—In this study, we outline a potential problem in normalising texts that are based on a modified version of the Arabic alphabet. One of the main resources available for processing resource-scarce languages is raw text collected from the Internet. Many less-resourced languages, such as Kurdish, Farsi, Urdu, Pashtu, etc., use a modified version of the Arabic writing system. Many characters in harvested data from the Internet may have exactly the same form but encoded with different Unicode values (ambiguous characters). The existence of ambiguous characters in words leads to word duplication, thus it is important to identify and unify ambiguous characters during the normalisation stage. Here, we demonstrate cases related to ambiguous Kurdish and Farsi characters and propose a semi-automatic approach to identifying and unifying them.**

Keywords-Kurdish; Sorani; Unicode; lexicography

I. INTRODUCTION

The main challenges in processing less-resourced languages is the lack of natural language processing (NLP) tools and resources.

Large numbers of languages, such as Kurdish, Farsi, Urdu, Pasthtu, etc., use a modified version of the Arabic writing system. We have observed that some characters of these languages have exactly the same form but are encoded differently. The problem with the inconsistent encoding of some characters (ambiguous characters) is that they are treated as different characters. This makes large numbers of similar words, which are similar in meaning and form, to be treated as completely different words. In this paper, we attempt to shed light on ambiguous characters, which results in generating multiple instances of words of similar forms but different encodings. Moreover, we will show an approach for identifying ambiguous characters and correcting them by appropriately unifying their Unicode values. In this work, we will mainly focus on Kurdish, but we will show the applicability of our work to other related languages such as Farsi.

The rest of the paper is organised as follows: in section II we highlight some of the general challenges in processing Kurdish. In section III we describe our dataset and in sections IV and V we describe our approach to identifying and unifying Unicode values of characters of equal form and pronunciation. In section VI we briefly describe the applicability of our approach to processing other related languages and we will conclude our paper in section VII.

II. THE CHALLENGES OF PROCESSING KURDISH

There are several dialects in Kurdish, such as Sorani, Kurmanji, Zazaki, Hawrami, Gorani, etc. However, the main two dialects are Sorani and Kurmanji. These two dialects differ in many ways, one of the main differences is the writing style. Sorani uses a modified version of Arabic while Kurmanji uses a modified version of Latin [2, 3, 4, 5, 7].

The use of a modified version of the Arabic alphabet poses an interesting challenge in processing Sorani text, which is identifying multiple Unicode values that are assigned to Arabic-based letters that have the same form and to appropriately unifying them. From Table I we can see that the letter {ه} constitutes one letter (H, h, E, e) which is pronounced as either /ha/ or /a/, depending on its location in a word. If it appears at the start of a word it forms {هـ} and if it appears in the middle it forms {ـهـ}. In both cases, it is pronounced as /ha/ but it may be assigned different Unicode values. If it appears at the end of a word it forms {ـه}, if it appears in isolation it forms {ه} but in both cases it constitutes /E/ or /e/. In addition to these two cases, in most electronic texts, it may appear as a zero-width non-joiner (zwnj) character, which prevents joining a character from its follower [2]. For example, in the word {بارهەڵگرەکە} (barHelgreke, "The goods carrier") it constitutes /H/ in the fourth position, it constitutes /e/ in the fifth position, and it constitutes a zwnj character in position nine in the word. For the same letter (i.e., the letter {ه}), different Unicode values are often used. For example, when it appeared in position four in the word its Unicode value was u06BE, but in some cases it is assigned u0647. When it appeared in position five and nine, its Unicode value is either u0647, u06BE, or u06D5. This inconsistent encoding makes large numbers of words lose their unique forms. Table II contains examples of different words that have the same form but different Unicode values. This kind of ambiguity has also been observed in Urdu [1, 6]. The problem that we are going to address is related to identifying ambiguous characters (i.e., characters that have the same form but different Unicode values) and unify their Unicode values. The solution to this problem is essential during the normalisation process of Sorani text because ignoring

this problem will lead to incorrectly treating large numbers of words as unique words.

The difficulty in processing Kurdish is further aggravated by the lack of gold-standard dataset. Although there are several dictionaries available for Kurdish annotated corpus but large datasets are still unavailable [5]. It is possible to use the large data that is available on the Internet for developing a corpus of raw text. However, due to the existence of ambiguous characters in the harvested data from the Internet, this problem should be solved during normalisation stage.

III. Dataset Collection

Fortunately, there is a large number of Kurdish news websites, where we can harvest data. We have collected various data from several websites. The collected dataset contains about 1,000,000 words, which is large enough to capture a large variety of word forms.

In the first step of the process we collected over 21,000 news articles from a large number of websites. Then, we parsed each web page and removed various unwanted data, such as mark-up text, numbers, punctuations, foreign words, etc. A small challenge in this step is that although it is easy to identify Latin-based scripts in the pages, detecting Arabic or Farsi words is hard because they share the same writing system as Kurdish Sorani. A simple way to tackle this issue is to extract all unique words from the data with a specific frequency threshold. We have intentionally removed words that have occurred less than 0.0001% in the data. These words were either Arabic or Farsi words, which are occasionally used in Kurdish news articles; words with incorrect spelling; and words that are accidentally merged with some other words during the parsing process of the web pages.

IV. Identifying unique characters

Once a set of clean text is retrieved we processed all the data and generated a lexicon, which contained unique words, and manually inspected the top 1000 most frequently occurred words. At this stage, a large number of words were treated as unique words even though they had similar forms and meanings with some other words. For example, as we have mentioned previously, some Arabic-based characters are ambiguous, these ambiguous characters may appear in many words that are exactly the same in terms of meaning and form. Table II contains examples of some of the most ambiguously occurring words in the lexicon. Also, we can note from Table II the frequency of most ambiguous words is high.

The identification of ambiguous characters in words is performed by manually inspecting the encoding values of characters in many frequently occurring words. Using the identification of unique words is time consuming and does not give an accurate account of the level of character ambiguity in the data and it is neither efficient nor easy to locate ambiguous characters in large numbers of words.

An efficiency improvement can be achieved by processing every character in every word in the lexicon and record all the unique characters along with their Unicode values. Then manually inspect the encoding of the recorded characters. However, the inefficiency aspect of this approach is it requires processing very large numbers of characters. For example, our dataset contained 1,983,579 words and the average word length was 6 characters, which yielded approximately 12 million characters to process. The time taken to process all the words was 56 seconds. This approach can be improved using a very simple technique. That is, recording all the unique words in a second lexicon. Then, process the characters of the recorded unique words. The total number of unique words was dramatically reduced to 42,987 words and the processing time was reduced to 26 seconds, which includes the time for creating the second lexicon.

TABLE I. Ambiguous Arabic-base characters

Total words	Frequency	Unicode value
که (ke, “as”)	125881	u0643 u06D5
که (ke, “as”)	92812	u0643 u0647
که (ke, “as”)	39747	u06A9 u0647
که (ke, “as”)	11312	u0643 u06D5
کوردستان (kurdistan, “Kurdistan”)	16081	u0643 u0648 u0631 u062F u0633 u062A u0627 0646
کوردستان (kurdistan, “Kurdistan”)	13196	u06A9 u0648 u0631 u062F u0633 u062A u0627 0646
ئیمه (aeme, “us”)	4252	u0626 u06CE u0645 u0647
ئیمه (aeme, “us”)	4050	u0626 u06CE u0645 u06D5
هێزی (hyz, “its power”)	2472	u0647 u06CE u0632 u06CC

TABLE II. Ambiguous words with their frequency and Unicode values

Unicode value	Latin-based letters	Arabic-based letters	Unicode value
u0048	H	ه	u06BE u06D5 or u0647
u0068	h		
u0049	I	-	-
u0069	i		
u0055	U	و	u0648
u0075	u		
u0057	W	و	u0648
u0077	w		
u0059	Y	ی	u06CC
u0079	y		

Once we identified all the unique characters and their Unicode, we then identified three ambiguous characters. Those characters are shown in Table III. The simple steps for finding the Unicode values of ambiguous characters are given in Fig. 1, which creates a list of all unique characters from all words for manual inspection.

V. UNIFYING DIFFERENT UNICODE VALUES OF SIMILAR CHARACTERS

Once we identified the ambiguous characters, we generated a mapping dictionary that mapped the Unicode value of ambiguous characters to a different Unicode value, which is shown in Table IV. The content of the mapping dictionary is simple and can be formatted in any style.

Generally, if we find a specific character with a specific Unicode value in a word then we replace it with a given (correct) Unicode value. However, as it can be noted from Table IV the Unicode value u0647, which represents {ه} (a, "a"), (h, "ha"), or zwnj should remain as it is or be mapped to u06BE or u06D5. The location in which the character appears dictates its form. If the character was followed by a character with the same Unicode value then it is changed to u06BE Unicode. Otherwise, there are exceptional cases for correctly mapping u0647 to u06BE or u06D5 Unicode values: (i) if the character is final then we replace it with u06D5. (ii) If a specific vowel (with the Unicode value u06CE, u06CC, u0627, or u06c6) follows the character then it should be mapped to u06D5. (iii) If the previous two cases do not apply then it should be mapped to u06BE.

The mapping dictionary, as shown in Fig. 2, that we have compiled contains one entry per line. Each entry contains a list of comma separated Unicode values (parameters), where the first parameter represents the Unicode value of an ambiguous character in a word and the second parameter is a list representing the Unicode value(s) that is used for replacing the ambiguous character. In order to deal with exceptional cases for handling u0647 Unicode value, the format of the dictionary entry for characters with u0647 Unicode value is the following: the first parameter is u0647 Unicode value; the first value in the list is the value that replaces u0647 if the character with u0647 Unicode value is a final character; the third parameter in the entry is a list of *n* number of Unicode values, where *n* is a positive number; the last parameter is the value that is used for replacing the Unicode value of the character with u0647 value if and only if the immediate following character is similar to the Unicode values in the list of *n* Unicode values.

From the list of characters that we have identified by following the steps in Fig. 1 we have manually inspected the characters that were of the same form but with different Unicode values. This way we have identified the characters that had the same form but different Unicode values, which resulted in duplicating a large number of words. Once we have identified all the ambiguous characters, we have compiled a mapping dictionary for replacing the Unicode values, which is shown in Table IV. The evaluation of the solution is conducted by extracting all the unique characters and their Unicode values from the lexicon and manually inspecting them to identify a character that is similar in form and pronunciation to one or more character(s) but with different Unicode value. The absence of an ambiguous character indicated that all characters in the lexicon were encoded correctly. It should be noted that the number of alphabet of any languages is not large and manual inspection of their Unicode values is not time consuming.

1. read words from a file and add them to a lexicon (*L*).
2. count the frequency of each word in *L* and create a create a new lexicon containing words and their frequency (*LF*).
   2.1. optional: sort content of *LF* in ascending order.
   2.2. for each word in *LF*, write the word and its Unicode values to a file for manual inspection.
   2.3. retrieve the characters of each word in *LF*.
      2.3.1. add the characters to a list (*CL*) if it is not in *CL*.
      2.3.2. write the character and their Unicode values to file if it is not in *CL*.
3. Inspect the characters that were written to the file in step 2.3.2 and identify n duplicate characters, where n is a predetermine number with the same form but with different Unicode values.

Figure 1. Outline of steps for identifying different Unicode values of characters of the same form

TABLE III. AMBIGUOUS CHARACTERS

Characters	Frequency	Unicode values
ه (pronounced as /ha/, /a/ and used as zero-width non-joiner character)	2361391	u06D5
	1961352	u0647
	51442	u06BE
ی (pronounced as /ye/)	2481987	u06CC
	69363	u0649
ک (pronounced as /k/)	585728	u06A9
	537621	u0643

TABLE IV. MAPPING BETWEEN UNICODE VALUES

Unicode value	Mapped Unicode value
u0647	u06BE or u06D5
u0649	u06CC
u0643	u06A9

VI. APPLYING OUR APPROACH TO RELATED LANGUAGES

We applied the same approach to Farsi, which is closely related to Kurdish. From our experiment on Farsi we identified that in Farsi the number of ambiguous characters are less than those in Kurdish. For example, from Table V we can see that the final

and medial characters {ي} (y, “y”) appear with different Unicode values. It is noticeable that the final {ي} (y, “y”) has u06CC assignment more frequently than u06BE while a medial {ـيـ} (y, “y”) is assigned u06CC Unicode value more than u06BE. Unlike in Kurdish, the character {ه} (a, “a”) have not been assigned the Unicode value u06BE. The u0647 Unicode value is assigned to the initial, medial and final character {ه} (a, “a”) more than u06D5 Unicode value. The third ambiguous character in Farsi was the character {ک} (k, “K”) which was often assigned the Unicode values u06A9 instead of u0643.

As shown in Table V those ambiguous characters neither change the semantic nor the form of the words but some NLP tools (such as text normalisers) treat words that contain ambiguous characters as different words, because of the differences in the Unicode values of some of their characters. In conclusion, after applying the same technique to related languages we could identify ambiguous characters and semi-automatically correct them.

VII. CONCLUSION

The normalisation of text often involves removing unwanted texts (noise) such as foreign words, numbers, punctuations, etc. This stage of text processing is one of the main stages in processing less-resourced languages because in most cases raw data is collected from the Internet, which may contain various noise. In addition to noise removal processing of online text we have identified an interesting case in processing Kurdish, and other related languages such as Farsi, where some characters of similar form and pronunciation are assigned different Unicode values (ambiguous characters). We anticipate that the reason is that for languages that use a modified version of Arabic script for writing may interchangeably use different Unicode values, which could be the Unicode value of the original Arabic character or a specific Unicode value for the modified character. Another possibility is that it may be due to the type of Operating Systems or the data entry devices that are used in producing the web pages, where they have different Unicode values for characters with similar forms.

Unifying ambiguous characters is an important step in the text normalisation stage because ambiguous characters, which are used for constructing words, lead to duplication of words. In many inductive NLP processing tasks it is not plausible to induce information from noisy data. Therefore, unifying Unicode values of ambiguous characters is an essential step towards removing noise.

In this paper, we have presented a semi-automatic approach to unifying Unicode values of Kurdish text. Furthermore, we have applied the same approach to Farsi. Our experiment on Farsi shows that our approach could be applicable to other related languages, such as Urdu and Pashtu, which we aim to apply it to them in the near future.

```
u'\u0647': [u'\u06be', [u'\u06ce', u'\u06d5', u'\u06cc'
u'\u0627' u'\u06c6'], u'\u06d5']
u'\u0643': [u'\u06a9']
u'\u0649': [u'\u06cc']
u'\u064a': [u'\u06cc']
```

Figure 2. Entries in the mapping-character file

TABLE V. FARSI AMBIGUOUS WORDS

Words	Frequency	Unicode value
آئین	118	u0622 u0626 u06CC u0646
آئین	10	u0622 u0626 u06BE u0646
آزادی	112	u0622 u0632 u0627 u062F u06CC
آزادی	11	u0622 u0632 u0627 u062F u0649
جامعه	197	u062C u0627 u0645 u0639 u0647
جامعه	22	u062C u0627 u0645 u0639 u06D5
حاکم	183	u062D u0627 u06A9 u0645
حاکم	14	u062D u0627 u0643 u0645

REFERENCES

[1] U. I. Bajwa, Z. Rehman, and W. Anwar, “Challenges in Urdu text tokenization and sentence boundary disambiguation,” Proc. 2nd Workshop on Southeast Asian Natural Language Processing, Chiang May, Thailand, 2011, pp. 40-45.

[2] K. S. Gautier, “Building a Kurdish language corpus: an overview of the technical problems,” Proc. 6th International Conference and Exhibition on Multilingual Computing (ICEMCO98), Cambridge, UK, April, 2012.

[3] D. N. MacKenzie, Kurdish dialect studies, volume 1 of London Oriental Series. Oxford University Press, 1961.

[4] E. N. McCarus, A Kurdish grammar descriptive analysis of the Kurdish of Sulaimaniya Iraq. PhD thesis. New York, USA: American Council of Learned Societies. 1958.

[5] G. B. Walther, and B. Sagot, “Developing a large-scale lexicon for a less-resourced language: general methodology and preliminary experiments on Sorani Kurdish,” Proc. 7th SaLTMiL Workshop on Creation and use of basic lexical resources for less-resourced languages (LREC 2010 Workshop), Valetta, Malta, 2010.

[6] M. Shamsfard, “Challenges and open problems in Persian text processing,” Proc. 5th Language and Technology Conference (LTC 2011), Poland, 2011, pp. 65-69.

[7] W. M. Thackston, Sorani Kurdish: A Reference Grammar with Selected Readings. Oxford University Press, UK. 1960.